Днес Python е един от най-използваните програмни езици и един от най-предпочитаните от програмистите. Той има интуитивен и разбираем синтаксис и върви с редица рамки, които да подсилят работата ти.

Винаги е била повсеместна дилема, когато става въпрос за избора на рамка за следващия ви проект. На всеки няколко месеца виждате нова технология и рамка, която преодолява слабостта на предишната, която сте използвали.

В мемента двете най използвани и популярни Python рамки са **Flask** и **Django** като и двете си има собствените си минуси и плюсове.

Но без значение коя от двете рамики ще изберете няма да сгрешите в никакъв случай

Първоначално стартиран през 2010 г. от Armin Ronacher, Flask е известен като микро-рамка, тъй като има малка или никаква зависимост от външни библиотеки. Използвайки го, разработчиците имат гъвкавостта да избират шаблони за проектиране, инструменти и бази данни. По този начин гъвкавостта е основната характеристика на тази Python рамка.

Тъй като индустрията за уеб разработка сега клони повече към микро-услуги и server-less платформи, популярността на Flask непрекъснато нараства. Той се използва широко за изграждане на мащабируеми уеб приложения без усилие. Всичко това благодарение на неговите уникални характеристики.

### **Features of Flask**

* Вграден сървър за разработка и бърз дебъгер
* Висока мащабируемост
* Използва шаблони Jinja2 и е съвместим с WSGI 1.0
* ORM-agnostic and highly flexible
* HTTP рекуести които подържат функционалноста
* Дава възможност за unit testing
* Лесен за работа

КАКВО Е Django

Стартиран през 2005 г., Django е рамка с пълен стек, проектирана с основната цел да улесни разработването на сложни уебсайтове, управлявани от база данни. Той улеснява сигурното и по-бързо разработване, като освобождава разработчиците от извършването на повтарящи се задачи за уеб разработка. С много отлични функции, той позволява на разработчиците да създават стабилни и високоефективни приложения.

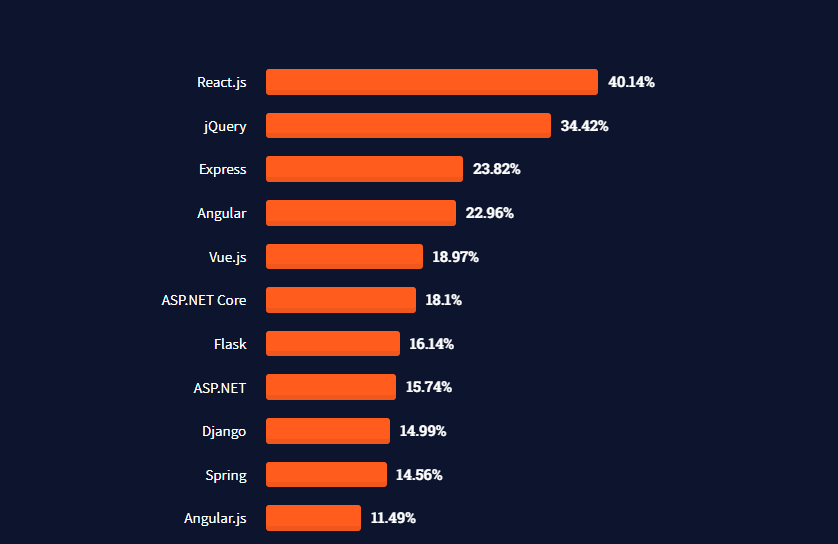
### **Features of Django**

* Позволява бързо разработка
* лесен за мащабиране и подържане
* Много сигурен и многофункционален
* Осигурява поддръжка за много front-end инструменти

Докато Flask и Django са рамки на Python, подходящи за разработване на уеб приложения, които са с висока производителност и лесни за мащабиране, те са много различни една от друга.

Макар Django да позволява бързо създаване на приложения той е сравнително по-бавен от Flask.

Микро-рамка Flask

Flask е т.нар. **micro-framework**, много лека микро-рамка, която контрастира с full-stack рамки като Django. Основната ѝ функционалност е малка, но е чувствително разширяема, правейки я подходяща за разработка както на малки, така и на големи приложения. В момента е една от най-използваните технологични рамки изобщо, по данни от анкетата на StackOverflow, и е в топ 3 на най-използваните рамки с Python. 

Едно от големите предимства на Flask е фактът, че е много **интуитивна рамка** и е лесна за използване, както от по-неопитни програмисти, така и от експерти. Известно е, че популярни компании като Netflix, LinkedIn, AirBnB и reddit използват framework-а в работата си.

Въпреки че е подходяща за начинаещи, Flask предлага всички функционалности, необходими за създаването на **back-end-а на добре работещо уеб приложение**. Тя идва с технологиите и инструментите, с които да създаваш необходимите ти функционалности, фокусирайки се върху тях, а не върху дизайна и визията на приложението.

Чрез нея можеш бързо и лесно да започнеш да разработваш уеб апликации, които могат да бъдат **скалирани**. Проектът ти, разработен с помощта на Flask, ще има изчистен и подреден код, без парчета за функционалности, които не се ползват и не са необходими на този етап.

Предвид това колко широкоразпространено е ползването на рамката, можеш да очакваш да попаднеш в **богата и активна общност** от разработчици. Именно на тях и базата от знания, които са развили и придобили, можеш да разчиташ, ако срещаш трудности или имаш въпроси за работата с Flask.

Макар и лека микро-рамка, Flask ти помага да разработваш както простички, така и мащабни **приложения с разнообразни цели**. С него можеш да създадеш блог, приложения като социални мрежи, метеорологични апликации, форми за събиране на обратна връзка, сайт за портфолио, с който да демонстрираш на потенциални работодатели, че имаш опит с рамката, а дори и да разработиш machine learning модели.

Когато имаш нужда от **гъвкавост** в проектите си и търсиш **възможности за персонализиране**, то рамката няма да те подведе. Особено ако става дума за **технологични решения**, които ще се отложат във времето, тези опции за персонализиране и промяна ще ти бъдат от голяма полза. Това по свой начин може да подпомогне производителността ти, така че **да ускориш процеса** по разработка.

Именно в това се крие и основното предимство пред цялостни, комплексни рамки като Django. Ще можеш бързо да създаваш уеб приложения, към които да добавяш нови и наистина необходими функционалности **на по-късен етап**.

За разлика от рамката на Django, Flask е много Pythonic. Лесно е да започнете с Flask, защото няма огромна крива на обучение.

Освен това е много ясен, което увеличава четливостта. За да създадете приложението „Hello World“, имате нужда само от няколко реда код.

from flask import Flask  
app = Flask(\_\_name\_\_)  
  
@app. route ('/')  
def hello world ():  
 return 'Hello World!'  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 apparent ()

Ако искате да се развивате на вашия локален компютър, можете да го направите лесно. Запазете тази програма като server.py и я стартирайте с python server.py.

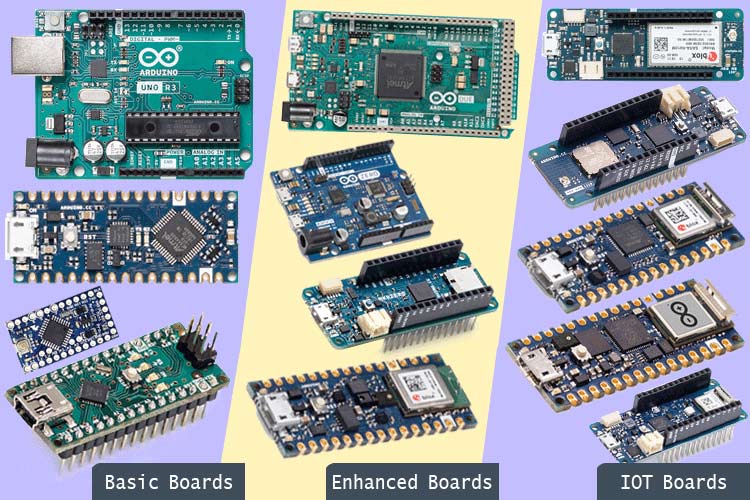
$ python server.py  
 \* Serving Flask app "hello"  
 \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

След това стартира уеб сървър, който е достъпен само на вашия компютър. В уеб браузър отворете localhost на порт :5000 (url) и ще видите да се показва „Hello World“.

За да хоствате и разработвате онлайн, можете да използвате PythonAnywhere

Какво е ардуино

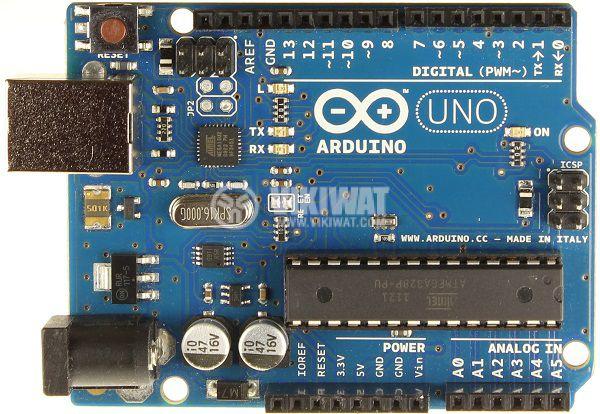
По същество [Arduino](https://www.arduino.cc/" \t "_blank) са **микроконтролерни развойни платки**. Arduino се състои от 8-битов Atmel AVR микроконтролер с допълващи се компоненти, които улесняват програмирането и включването в други вериги. Важен аспект на Arduino платформата, е наличието на стандартни конектори, които позволяват на потребителите да свързват CPU платката към голям набор от различни, взаимнозаменяеми модули, наречени шилдове. Важно е да отбележим, че Arduino е хардуер с отворен код.



**А сега на по-прост език**: Arduino платките са като мини-компютри, които може да програмирате да правят различни неща и да взаимодействат със света посредством електронни сензори, светодиоди и електромоторчета. Всъщност чрез тези платки, сериозни проекти свързани с електроника вече да достъпни до всички. Дори хора на изкуството и креативността, дизайнери, могат да превръщат идеите си в реалност.

Arduino микроконтролерите са подходящи и за всеки с хоби или интерес да създава интерактивни обекти. Звучи интересно, нали?

Ардуино представляват микроконтролерни развойни платки с отворен код, базирани на принципа „лесен за използване“ хардуер и софтуер. Те взаимодействат с външното им обкръжение чрез различни сензори, бутони, електромотори, светодиоди и други, което позволява на разработчиците на програми да създават широк набор от приложения. Голямо предимство на Ардуино е, че конекторите им са стандартни (като например USB), което позволява лесно свързване с други устройства и системи.



Важно от финансова гледна точка е да отбележим, че тези електронни платки могат както да се закупят готови, така и да бъдат сглобени от потребители с достатъчно познания в тази област. [Серията Ардуино](https://vikiwat.com/manufacturer/276/arduino.html) включва микроконтролери и процесори на различни производители – Atmel, ARM, Intel. Управлението на платките става посредством набор от инструкции на програмния език за програмиране [Arduino](https://www.arduino.cc/en/Reference/HomePage)(базиран на Wiring) и средата за разработка [Arduino Software](https://www.arduino.cc/en/Main/Software) (базирана на Processing).

Ардуино възниква като платформа за разработка на проекти от студенти без особен опит в областта на електрониката и програмирането, но впоследствие, след като достига по-широк кръг от потребители и набира голяма популярност, тя започва да се използва в много по-сложни проекти в сфери като IoT, 3D принтиране, вградени системи и др. През годините разработчици от цял свят  (студенти, любители, артисти, програмисти, специалисти в различни области) допринасят за натрупването на огромен обем информация и проекти, които са със свободен достъп и помагат в работата, както на хора, които тепърва навлизат в света на Ардуино, така и на експерти.

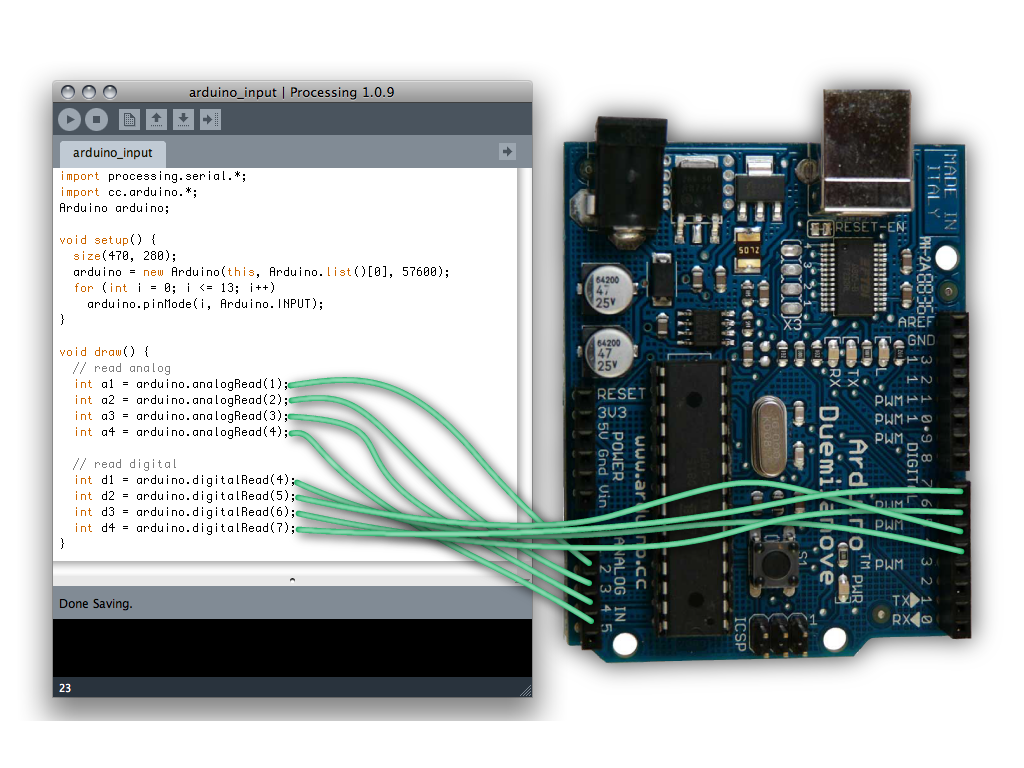
**Основните предимства на тази ардуино са следните:**

* ниска цена – струва не повече от 50 долара;
* представлява крос платформа – средата за разработка функционира без проблем както под Windows, така и под Macintosh OSX и Linux;
* опростена, изчистена програмна среда;
* разширяем софтуер с отворен код;
* разширяем хардуер с отворен код.

**Проектите за Ардуино се създават чрез средата за разработка Arduino Software**

Ако разполагаме с надеждна Интернет връзка, е най-добре да използваме онлайн варианта на средата – Arduino Web Editor. Той ни позволява да съхраняваме проектите си в глобално Интернет пространство (облак), правейки ги достъпни от всяко друго устройство и дава възможност да имаме резервни копия.

Основно предимство на тази опция е и това, че винаги работим с най-новата версия на средата, без да има нужда да инсталираме ъпдейти или допълнителни библиотеки. Всичко, от което се нуждаем в този случай, е да си създадем потребителски акаунт на [auth.arduino.cc/register](https://auth.arduino.cc/register), да отидем на [arduino.cc/editor](https://create.arduino.cc/editor) и да добавим един плъгин към нашия браузър.



Сега вече можем да влезем в акаунта си и да пишем и качваме проекти на Ардуино платките, с които разполагаме. Уеб средата разпознава автоматично свързаните към персоналния ни компютър Ардуино платформи и се самонастройва в зависимост от конкретния вид на платката.



За да проверим, че всичко е наред, можем да стартираме един от предварително готовите проекти, които ни предоставя средата. Това става като отидем на меню ‘Examples’, изберем ‘Basic’ и след това ‘Blink’. Така проектът се зарежда в пространството за писане на код. След това зареждаме проекта на Ардуино платката чрез бутона ‘Upload’ и резултатът, който трябва да видим, е жълтият светодиод на платформата да пресветва през една секунда в безкраен цикъл.

Ардуино има множество модули и платки, но тойзи който ще позлваме в този проек е esp8266

ESP8266 e един от най големите скокове в мирко контролерите, заради вградения wifi модул в него, до сега проектите бяха изолирани и не можеха да бътат свързвани в мрежа и по този начин да могат да наблюдават или управляват проекта от Интернет навсякъде на планетата.



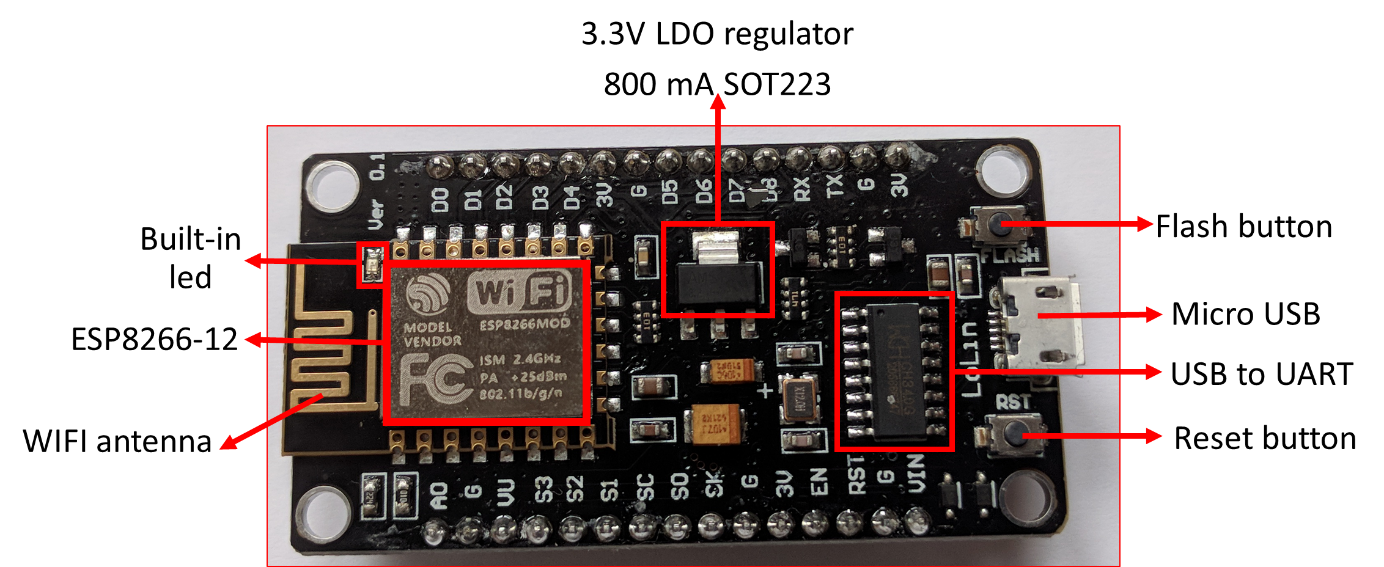
Малко история

Първата компания, създала това **Чипът ESP8266 беше Espressif,** китайска компания, разположена в Шанхай, въпреки че в момента има и други производители, които я разработват и произвеждат. Точната дата на пускането му беше през лятото на 2014 г., така че не е толкова стара. Започва да се предлага на пазара на ниска цена и това заедно с възможностите му го правят много популярен скоро.



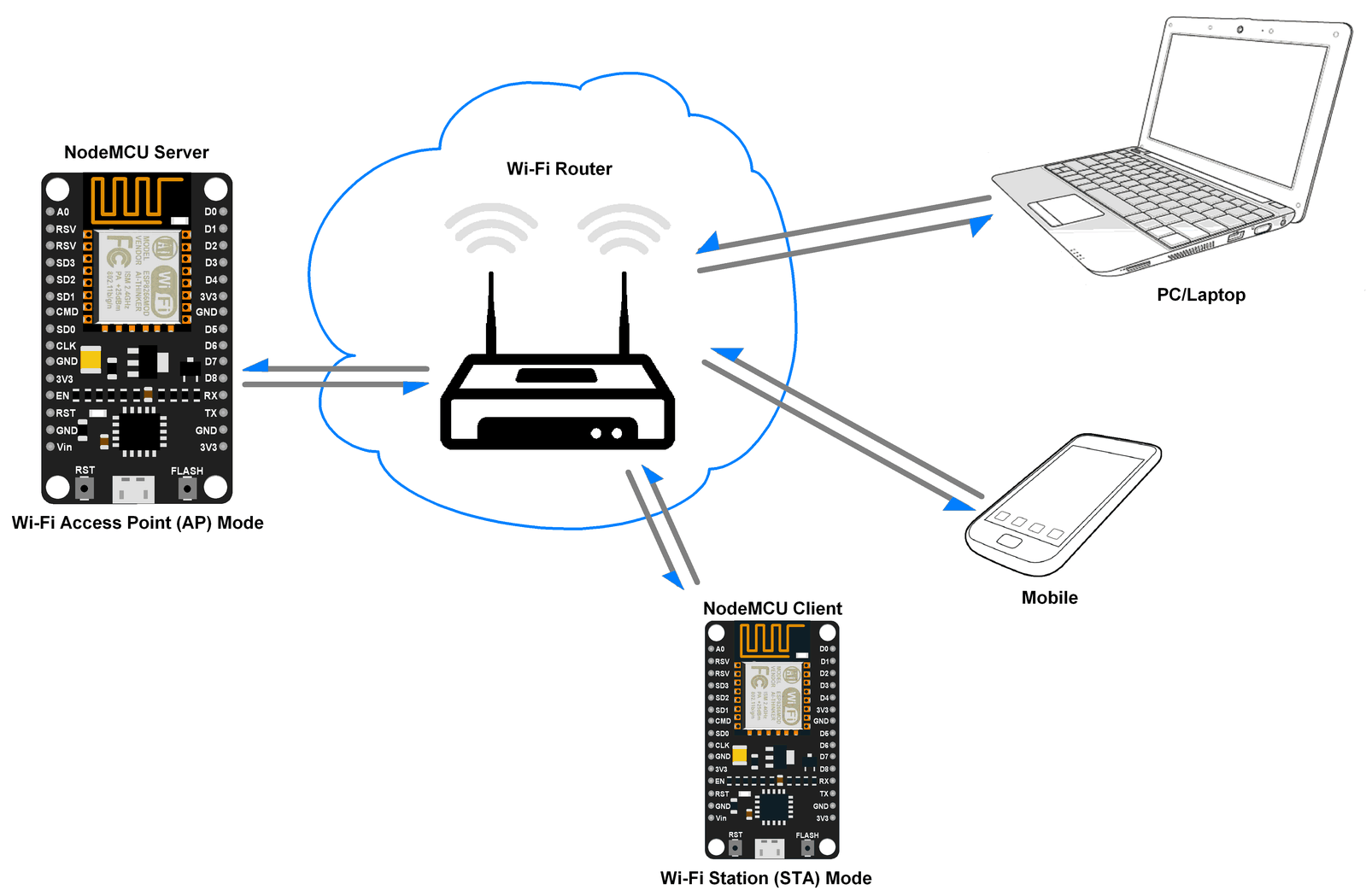
La**общност за разработчици** той също имаше важна роля за успеха, тъй като те започнаха да превеждат и публикуват голямо количество документация, да създават фърмуери и други кодове, които да се използват на ESP8266. Това даде на производителите всички необходими инструменти, за да могат да използват устройството с пълната си мощ.

Но трябва да знаете, че както при транзисторите,**номенклатурата или номерирането** Не винаги е бил ESP8266, но първо първо се появяват някои ранни ESP, след това идват версии като ESP8285 от 2016 г., които включват интегрирана 1MB памет flahsd, а след това ще се появи ESP8266,



който днес знаем, който изглежда е направил крачка назад тъй като не разполага с такава памет, но други външни чипове могат да се добавят за съхранение на програми.

**ESP8266 може да се интегрира в WiFi** който осигурява евтин чип с пълен TCP / IP стек и микроконтролер. Той се захранва от 3.3v и има 106 Mhz Tensilica Xtensa LX80 процесор, 64 KB RAM за инструкции и 96 KB за данни, 16 GPIO пина, специални UART пинове и SPI и I2C интерфейс.



La **Процесор Tensilica** може да се направи по-бързо чрез овърклок, който някои, но не всички модели позволяват. Всъщност тактовата честота може да се удвои. Между другото, 32-битов RISC тип процесор. Също така в модула е включен 10-битов ADC конвертор за сигналите.

Като допълнение, той включва външен чип QSPI флаш памет от 512 KB до 4 MB в зависимост от модула, понякога дори може да достигне 16 MB. Относно **Възможности за WiFi свързаност**, той е съвместим със стандарта IEEE 802.11 b / g / n, в допълнение към поддържането на защитата WEP, WPA и WPA2.

**Какво е PostgreSQL**

PostgreSQL е мощна обектно-релационна база данни с отворен код, която използва и разширява езика SQL, комбиниран с много функции, които безопасно съхраняват и мащабират най-сложните работни натоварвания с данни. Произходът на PostgreSQL датира от 1986 г. като част от проекта POSTGRES в Калифорнийския университет в Бъркли и има повече от 30 години активно развитие на основната платформа.BSD лицензирана обектно релационна база данни. (ORDBMS)Пионер, чиито концепции по-късно са се появили в комерсиалните бази данни.



PostgreSQL спечели силна репутация със своята доказана архитектура, надеждност, интегритет на данните, стабилен набор от функции, разширяемост и отдадеността на общността с отворен код зад софтуера за постоянно предоставяне на ефективни и иновативни решения. PostgreSQL работи на всички основни операционни системи, съвместим е с ACID от 2001 г. и има мощни добавки като популярния разширител на геопространствени бази данни PostGIS. Не е изненадващо, че PostgreSQL се превърна в предпочитаната релационна база данни с отворен код за много хора и организации.

**история**

1977-1985 Базиран на проекта POSTGRES в Бъркли

1986-1993 Трансформиран е в обектно ориентирана релационна база данни (ORDBMS)

1995 Добавена е поддръжка на SQL и проектът е преименуван на Postgres95

1996 Преименува се на PostgreSQL и продължава да се разработва като свободен софтуер

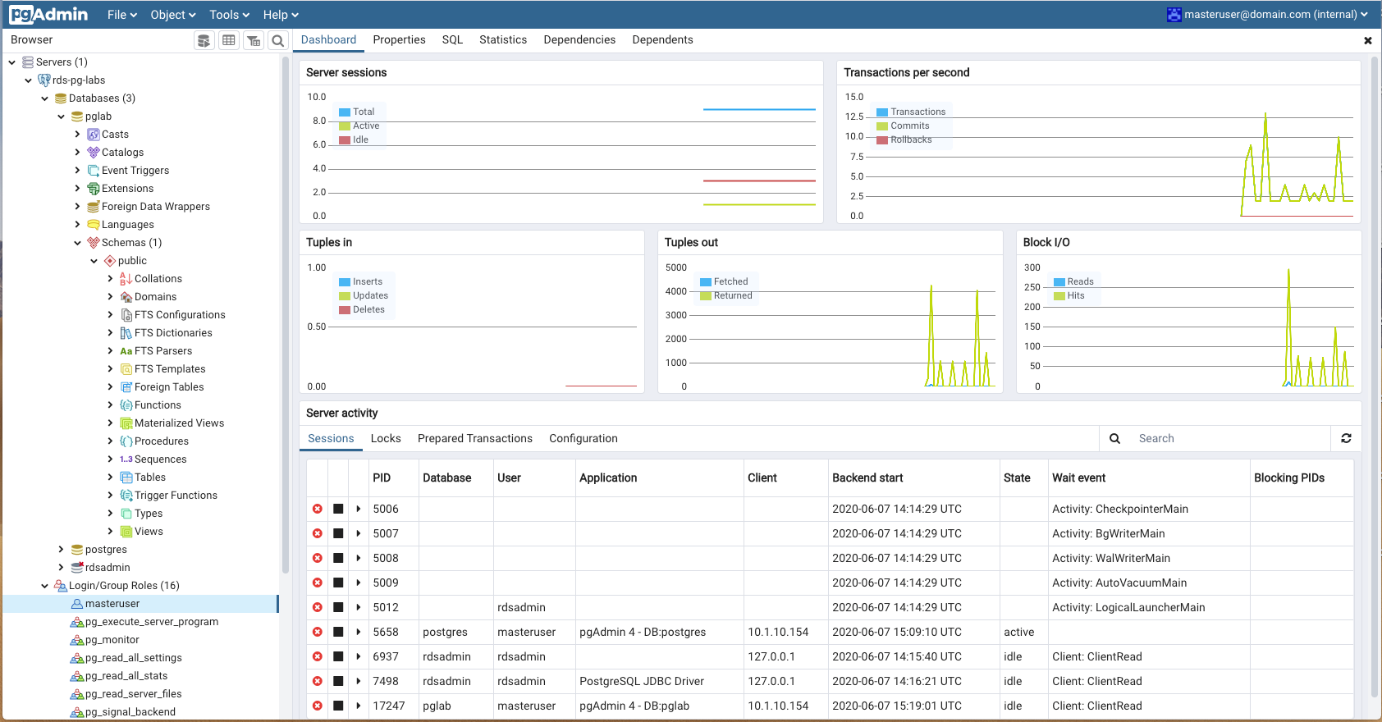
2000 Излиза версия 7.0 с поддръжка на FK, много подобрения в оптимизатора на заявки, бързодействието и много други

2000-2004 Излизат версии 7.1, 7.2, 7.3 и 7.4, като във всяка има много подобрения спрямо предишната

2005 Излиза версия 8.0 с вградена поддръжка на Windows, Savepoints, PITR, Table spaces и много други

Защо PostgreSQL

PostgreSQL идва с много функции, които имат за цел да помогнат на разработчиците да създават приложения, на администраторите да защитят целостта на данните и да изградят устойчиви на грешки среди и да ви помогнат да управлявате данните си, независимо колко голям или малък е наборът от данни. Освен че е безплатен и с отворен код, PostgreSQL е много разширим. Например, можете да дефинирате свои собствени типове данни, да създавате персонализирани функции, дори да пишете код от различни езици за програмиране, без да компилирате отново вашата база данни!



PostgreSQL се опитва да се съобрази със стандарта SQL, когато такова съответствие не противоречи на традиционните характеристики или може да доведе до лоши архитектурни решения. Много от функциите, изисквани от стандарта SQL, се поддържат, макар и понякога с леко различен синтаксис или функция.

С течение на времето могат да се очакват по-нататъшни стъпки към съответствие. Считано от изданието версия 10 през октомври 2017 г., PostgreSQL отговаря на поне 160 от 179 задължителни характеристики за съответствие с SQL:2011 Core, където към момента на писане на това никоя релационна база данни не отговаря на пълното съответствие с този стандарт.

Виодеве данни

* Primitives: Integer, Numeric, String, Boolean
* Structured: Date/Time, Array, Range, UUID
* Document: JSON/JSONB, XML, Key-value (Hstore)
* Geometry: Point, Line, Circle, Polygon
* Customizations: Composite, Custom Types

Целостта на данните

* UNIQUE, NOT NULL
* Primary Keys
* Foreign Keys
* Exclusion Constraints
* Explicit Locks, Advisory Locks

Сигурнсот

* Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more
* Robust access-control system
* Column and row-level security

Надеждност и възстановяване при проблем

* Write-ahead Logging (WAL)
* Replication: Asynchronous, Synchronous, Logical
* Point-in-time-recovery (PITR), active standbys
* Tablespaces

По всяко време PostgreSQL поддържа журнал за предварително записване (WAL) в поддиректорията pg\_wal/ на директорията с данни на клъстера. Дневникът записва всяка промяна, направена във файловете с данни на базата данни. Този дневник съществува предимно за целите на безопасността при сривове: ако системата се срине, базата данни може да бъде възстановена до последователност чрез „възпроизвеждане“ на записите в дневника, направени след последната контролна точка. Съществуването на регистрационния файл обаче прави възможно използването на трета стратегия за архивиране на бази данни: можем да комбинираме архивиране на ниво файлова система с архивиране на WAL файловете.



Ако е необходимо възстановяване, ние възстановяваме резервното копие на файловата система и след това възпроизвеждаме от архивираните WAL файлове, за да доведем системата до текущо състояние. Този подход е по-сложен за администриране от който и да е от предишните подходи, но има някои значителни предимства:

* Не се нуждаем от идеално последователно архивиране на файловата система като отправна точка. Всяко вътрешно несъответствие в резервното копие ще бъде коригирано чрез повторно възпроизвеждане на журнал (това не се различава значително от това, което се случва по време на възстановяване от срив). Така че не се нуждаем от възможност за моментна снимка на файловата система, а само от tar или подобен инструмент за архивиране.
* Тъй като можем да комбинираме безкрайно дълга поредица от WAL файлове за повторно възпроизвеждане, непрекъснато архивиране може да се постигне просто като продължим да архивираме WAL файловете. Това е особено ценно за големи бази данни, където може да не е удобно да се прави често пълно архивиране.
* Не е необходимо да възпроизвеждате WAL записите до края. Можем да спрем повторението във всеки момент и да имаме последователна моментна снимка на базата данни, каквато беше по това време. По този начин тази техника поддържа възстановяване в даден момент: възможно е да възстановите базата данни до нейното състояние по всяко време, откакто е направено вашето основно архивиране.
* Ако непрекъснато захранваме поредицата от WAL файлове на друга машина, която е заредена със същия основен архивен файл, имаме система за топъл режим на готовност: във всеки момент можем да изведем втората машина и тя ще има почти актуално копие на базата данни.

Както при обикновената техника за архивиране на файлова система, този метод може да поддържа възстановяване само на цял клъстер от база данни, а не на подмножество(subset). Освен това изисква много архивно съхранение: базовото архивиране може да е обемисто и натоварената система ще генерира много мегабайти WAL трафик, които трябва да бъдат архивирани. Все пак това е предпочитаната техника за архивиране в много ситуации, когато е необходима висока надеждност.

За да възстановите успешно с помощта на непрекъснато архивиране (наричано още „онлайн архивиране“ от много доставчици на база данни), имате нужда от непрекъсната поредица от архивирани WAL файлове, която се простира назад поне до началния час на вашето архивиране. Така че, за да започнете, трябва да настроите и тествате вашата процедура за архивиране на WAL файлове, преди да направите първото си базово архивиране.

Има недостатъците при възстановяване са няколко ограничения за непрекъснато архивиране. Но те вероятно ще бъдат коригирани в бъдещи версии:

* Командите CREATE TABLESPACE се записват в WAL с литерален абсолютен път и следователно ще бъдат възпроизвеждани като създаване на таблично пространство със същия абсолютен път. Това може да е нежелателно, ако регистрационният файл се възпроизвежда на друга машина. Може да бъде опасно, дори ако регистрационният файл се възпроизвежда на същата машина, но в нова директория с данни: повторното възпроизвеждане пак ще презапише съдържанието на оригиналното таблично пространство. За да избегнете потенциални проблеми от този вид, най-добрата практика е да направите ново базово архивиране след създаване или премахване на таблични пространства.
* Ако се изпълни команда CREATE DATABASE, докато се прави базово архивиране и след това базата данни на шаблона, копирана от CREATE DATABASE, е модифицирана, докато базовото архивиране все още е в ход, възможно е възстановяването да причини тези модификации да бъдат разпространени в създаде и база данни. Това разбира се е нежелателно. За да избегнете този риск, най-добре е да не променяте шаблонни бази данни, докато правите базово архивиране.

Трябва също да се отбележи, че форматът WAL по подразбиране е доста обемист, тъй като включва много моментни снимки на страници на диска. Тези моментни снимки на страници са предназначени да поддържат възстановяване при срив, тъй като може да се наложи да коригираме частично записани дискови страници. В зависимост от вашия системен хардуер и софтуер, рискът от частични записи може да е достатъчно малък, за да бъде игнориран, в който случай можете значително да намалите общия обем на архивираните регистрационни файлове, като изключите моментните снимки на страници с помощта на параметъра full\_page\_writes.

Изключването на моментни снимки на страници не предотвратява използването на регистрационните файлове за PITR операции. Област за бъдещо развитие е компресиране на архивирани WAL данни чрез премахване на ненужни копия на страници, дори когато full\_page\_writes е включен. Междувременно администраторите може да пожелаят да намалят броя на моментните снимки на страници, включени в WAL, като увеличат параметрите на интервала на контролните точки, доколкото е възможно.

https://www.postgresql.org/docs/